Introduction of Storage Services, Hadoop & Mapreduce

Portions of this PPT draw from PPT authored by Professor Dijiang Huang at Arizona State University

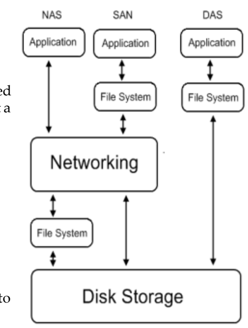
Agenda

- Storage Services
- Concepts of Mapreduce/Hadoop
- Hadoop 2.0 and YARN
- References
  - "Data-Intensive Text Processing with MapReduce" by Jimmy Lin and Chris Dyer, University of Maryland, College Park
  - Internet

Storage Services

Introduction

- Common storage architecture :
  - DAS - Direct Attached Storage
    - o Storage device was directly attached to a server or workstation, without a storage network in between.
  - NAS - Network Attached Storage
    - o File-level computer data storage connected to a computer network providing data access to heterogeneous clients.
  - SAN - Storage Area Network
    - o Attach remote storage devices to servers in such a way that the devices appear as locally attached to the operating system.

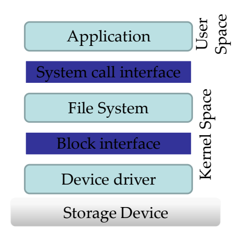


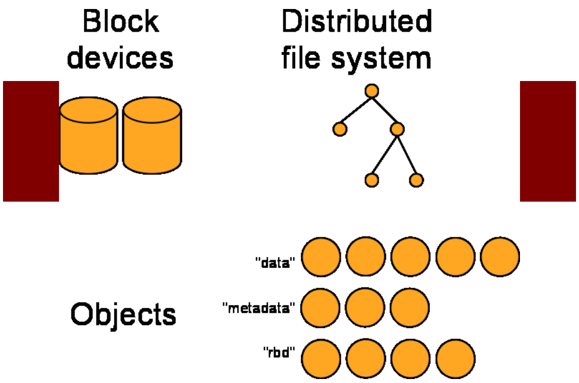
File System Level

- Data and Files
  - What is data ?
    - o Data is information that has been converted to a machine-readable, digital binary format.
    - o Control information indicates how data should be processed.
    - o Applications may embed control information in user data for formatting or presentation.
    - o Data and its associated control information is organized into discrete units as files or records.
  - What is file ?
    - o Files are the common containers for user data, application code, and operating system executables and parameters.
  - Metadata
    - o The control information for file management is known as metadata.
    - o File metadata includes file attributes and pointers to the location of file data content.
    - o File metadata may be segregated from a file's data content.
    - o Metadata on file ownership and permissions is used in file access.
    - o File timestamp metadata facilitates automated processes such as backup and life cycle management.

What To Be Virtualized

- Layers can be virtualized
  - File system
    - o Provide compatible system call interface to user space applications.
  - Block device
    - o Provide compatible block device interface to file system.
    - o Through the interface such as
      - – SCSI (Small Computer System Interfaces)
      - – SAS (Serial Attached SCSI)
      - – ATA (a.k.a., IDE)
      - – SATA (Serial ATA)
      - – etc.





GFS: Assumptions

- Commodity hardware over "exotic" hardware
  - Scale "out", not "up"
- High component failure rates
  - Inexpensive commodity components fail all the time
- "Modest" number of huge files
  - Multi-gigabyte files are common, if not encouraged
- Files are write-once, mostly appended to
  - Perhaps concurrently
- Large streaming reads over random access
  - High sustained throughput over low latency

GFS: Design Decisions

- Files stored as chunks
  - Fixed size (64MB)
- Reliability through replication
  - Each chunk replicated across 3+ chunkservers
- Single master to coordinate access, keep metadata
  - Simple centralized management
- No data caching
  - Little benefit due to large datasets, streaming reads
- Simplify the API
  - Push some of the issues onto the client (e.g., data layout)

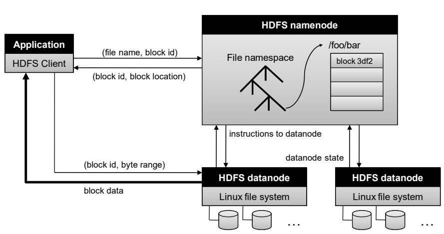
HDFS = GFS clone (same basic ideas)

From GFS to HDFS

- Terminology differences:
  - GFS master = Hadoop namenode
  - GFS chunkservers = Hadoop datanodes
- Functional differences:
  - No file appends in HDFS (planned feature)
  - HDFS performance is (likely) slower

For the most part, we'll use the Hadoop terminology…

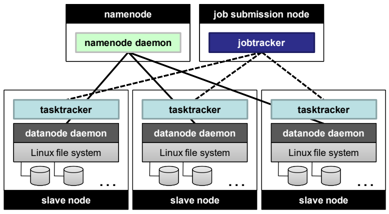
HDFS Architecture



Namenode Responsibilities

- Managing the file system namespace:
  - Holds file/directory structure, metadata, file-to-block mapping, access permissions, etc.
- Coordinating file operations:
  - Directs clients to datanodes for reads and writes
  - No data is moved through the namenode
- Maintaining overall health:
  - Periodic communication with the datanodes
  - Block re-replication and rebalancing
  - Garbage collection

Putting everything together…

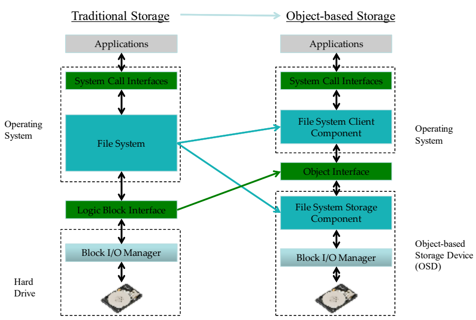


Ceph Design Goals

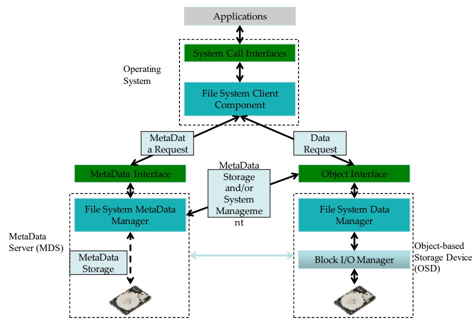
http://ceph.com/

- Ceph replicates data and makes it fault-tolerant, using commodity hardware and requiring no specific hardware support. As a result of its design, the system is both self-healing and self-managing, aiming to minimize administration time and other costs.
- Scalability
  - Storage capacity, throughput, client performance. Emphasis on HPC.
- Reliability
  - "…failures are the norm rather than the exception…"
- Performance
  - Dynamic workloads

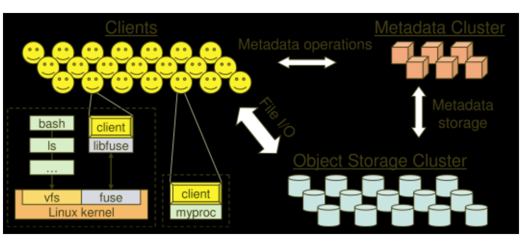
Object Based Storage: Separate the File



Object Based Storage: Create MetaData



System Overview



Key Features

- Decoupled data and metadata
  - CRUSH (Controlled Replication Under Scalable Hashing)
    - o It is an algorithm that determines how to store and retrieve data by computing data storage locations.
      - – Files striped onto predictably named objects
      - – CRUSH maps objects to storage devices
- Dynamic Distributed Metadata Management
  - Dynamic subtree partitioning
    - o Distributes metadata amongst MDSs
- Object-based storage
  - OSDs handle migration, replication, failure detection
  - ry

Client Operation

- Ceph interface
  - Nearly POSIX (Portable Operating System Interfaces)
  - Decoupled data and metadata operation
- User space implementation
  - FUSE (File system in USErspace) or directly linked

Client Access Example

1. Client sends *open* request to MDS
2. MDS returns capability, file inode, file size and stripe information
3. Client read/write directly from/to OSDs
4. MDS manages the capability
5. Client sends *close* request, relinquishes capability, provides details to MDS

Synchronization

- Adheres to POSIX
- Includes HPC oriented extensions
  - Consistency / correctness by default
  - Optionally relax constraints via extensions
  - Extensions for both data and metadata
- Synchronous I/O used with multiple writers or mix of readers and writers

Distributed Metadata

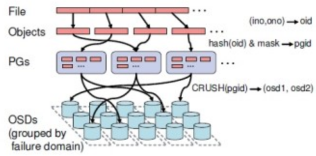
- "Metadata operations often make up as much as half of file system workloads…"
- MDSs use journaling
  - Repetitive metadata updates handled in memory
  - Optimizes on-disk layout for read access
- Adaptively distributes cached metadata across a set of nodes

Distributed Object Storage

- Files are split across objects
- Objects are members of placement groups
- Placement groups are distributed across OSDs.

Separating Data and Metadata

- **Data Distribution with CRUSH**
  - In order to avoid imbalance (OSD idle, empty) or load asymmetries (hot data on new device).
    →distributing new data randomly.
  - Use a simple hash function, Ceph maps objects to Placement groups (PGs). PGs are assigned to OSDs by CRUSH.



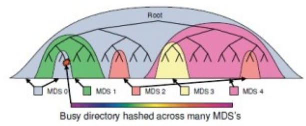
CRUSH

- CRUSH(x) → (osd_{n1}, osd_{n2}, osd_{n3})
  - Inputs
    - x is the placement group
    - Hierarchical cluster map
    - Placement rules
  - Outputs a list of OSDs
- Advantages
  - Anyone can calculate object location
  - Cluster map infrequently updated

Dynamic Distributed Metadata Management

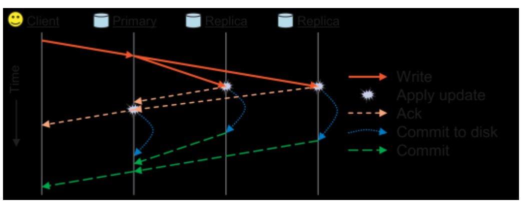
Dynamic Distributed Metadata Management

- Ceph utilizes a metadata cluster architecture based on Dynamic Subtree Partitioning.(workload balance)
- Dynamic Subtree Partitioning
  - Most FS ,use static subtree partitioning
    →imbalance workloads.
    →simple hash function can get directory.
  - Ceph's MDS cluster is based on a dynamic subtree partitioning. →balance workloads



Replication

- Objects are replicated on OSDs within same PG
  - Client is oblivious to replication



Failure Detection and Recovery

- *Down* and *Out*
- Monitors check for intermittent problems
- New or recovered OSDs peer with other OSDs within PG

Related Links for Ceph

- OBFS: A File System for Object-based Storage Devices
  - ssrc.cse.ucsc.edu/Papers/wang-mss04b.pdf
- OSD
  - www.snia.org/tech_activities/workgroups/osd/
- Ceph Presentation
  - http://institutes.lanl.gov/science/institutes/current/ComputerScience/ISSDM-07-26-2006-Brandt-Talk.pdf

Acronyms of Ceph

- **CRUSH**: Controlled Replication Under Scalable Hashing
- **EBOFS**: Extent and B-tree based Object File System
- **HPC**: High Performance Computing
- **MDS**: MetaData server
- **OSD**: Object Storage Device
- **PG**: Placement Group
- **POSIX**: Portable Operating System Interface for uniX
- **RADOS**: Reliable Autonomic Distributed Object Store

Hadoop Terminology

Google calls it:	Hadoop equivalent:
MapReduce	Hadoop
GFS	HDFS
Bigtable	HBase
Chubby	Zookeeper

Mapreduce

What is MapReduce?

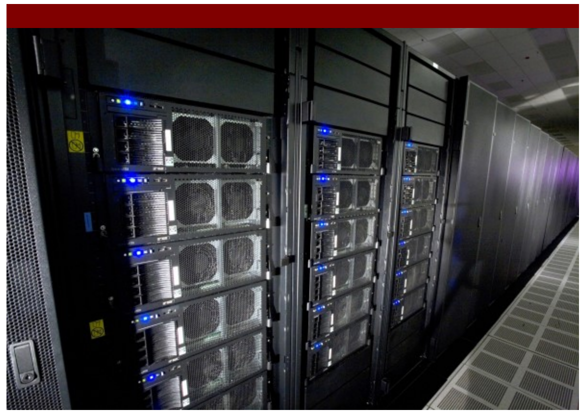
- Programming model for expressing distributed computations at a massive scale
- Execution framework for organizing and performing such computations
- Open-source implementation called Hadoop

Utility Computing

- What?
  - Computing resources as a metered service ("pay as you go")
  - Ability to dynamically provision virtual machines
- Why?
  - Cost: capital vs. operating expenses
  - Scalability: "infinite" capacity
  - Elasticity: scale up or down on demand
- Does it make sense?
  - Benefits to cloud users
  - Business case for cloud providers

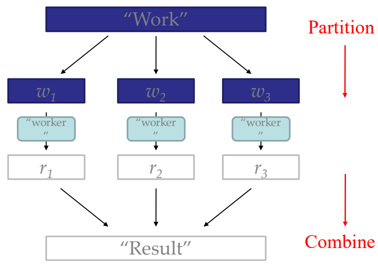
Cloud Resources

- Hadoop on your local machine
- Hadoop in a virtual machine on your local machine
- Hadoop in the Virtual Laboratory at CIDSE



Source: Wikipedia (IBM Roadrunner)

Divide and Conquer



Parallelization Challenges

- How do we assign work units to workers?
- What if we have more work units than workers?
- What if workers need to share partial results?
- How do we aggregate partial results?
- How do we know all the workers have finished?
- What if workers die?

What is the common theme of all of these problems?

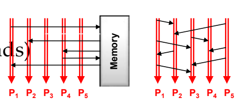
Common Theme?

- Parallelization problems arise from:
  - Communication between workers (e.g., to exchange state)
  - Access to shared resources (e.g., data)
- Thus, we need a synchronization mechanism

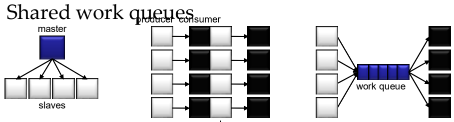
Managing Multiple Workers

- Difficult because
  - We don't know the order in which workers run
  - We don't know when workers interrupt each other
  - We don't know the order in which workers access shared data
- Thus, we need:
  - Semaphores (lock, unlock)
  - Conditional variables (wait, notify, broadcast)
  - Barriers
- Still, lots of problems:
  - Deadlock, livelock, race conditions...
  - Dining philosophers, sleeping barbers, cigarette smokers...
- Moral of the story: be careful!

- Programming models
  - Shared memory (pthreads)
  - Message passing (MPI)



- Design Patterns
  - Master-slaves
  - Producer-consumer flows
  - Shared work queues



Where the rubber meets the road

- Concurrency is difficult to reason about
- Concurrency is even more difficult to reason about
  - At the scale of datacenters (even across datacenters)
  - In the presence of failures
  - In terms of multiple interacting services
- Not to mention debugging…
- The reality:
  - Lots of one-off solutions, custom code
  - Write you own dedicated library, then program with it
  - Burden on the programmer to explicitly manage everything

What's the point?

- It's all about the right level of abstraction
  - The von Neumann architecture has served us well, but is no longer appropriate for the multi-core/cluster environment
- Hide system-level details from the developers
  - No more race conditions, lock contention, etc.
- Separating the *what* from *how*
  - Developer specifies the computation that needs to be performed
  - Execution framework ("runtime") handles actual execution

The datacenter *is* the computer!

"Big Ideas"

- Scale "out", not "up"
  - Limits of Symmetric Multiprocessing (SMP) and large shared-memory machines
- Move processing to the data
  - Cluster have limited bandwidth
- Process data sequentially, avoid random access
  - Seeks are expensive, disk throughput is reasonable
- Seamless scalability
  - From the mythical man-month to the tradable machine-hour

MapReduce

Typical Large-Data Problem

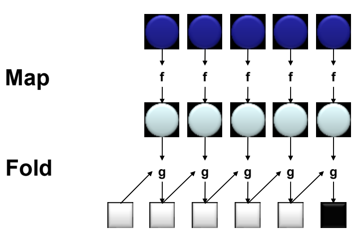
- Iterate over a large number of records
- Extract something of interest from each
- Shuffle and sort intermediate results
- Aggregate intermediate results
- Generate final output

Map

Reduce

Key idea: provide a functional abstraction for these two operations

MapReduce



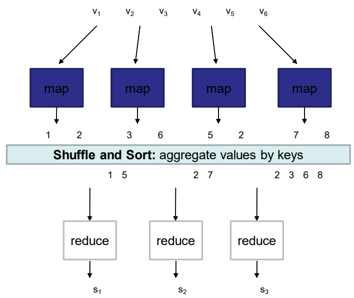
MapReduce

- Programmers specify two functions:

  map $(k_1, v_1) \rightarrow [(k_2, v_2)]$ denote a list

  reduce $(k_2, [v_2]) \rightarrow [(k_3, v_3)]$
  - All values with the same key are sent to the same reducer
- The execution framework handles everything else…

MapReduce



MapReduce

- Programmers specify two functions:
  - **map** $(k_1, v_1) \rightarrow [(k_2, v_2)]$
  - **reduce** $(k_2, [v_2]) \rightarrow [(k_3, v_3)]$
    - All values with the same key are sent to the same reducer
- The execution framework handles everything else…

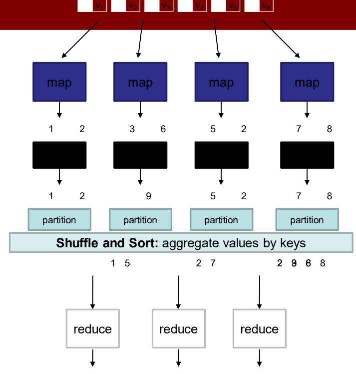
What's "everything else"?

MapReduce "Runtime"

- Handles scheduling
  - Assigns workers to map and reduce tasks
- Handles "data distribution"
  - Moves processes to data
- Handles synchronization
  - Gathers, sorts, and shuffles intermediate data
- Handles errors and faults
  - Detects worker failures and restarts
- Everything happens on top of a distributed FS (later)

MapReduce

- Programmers specify two functions:
  - **map** $(k_1, v_1) \rightarrow [(k_2, v_2)]$
  - **reduce** $(k_2, [v_2]) \rightarrow [(k_3, v_3)]$
    - All values with the same key are reduced together
- The execution framework handles everything else…
- Not quite…usually, programmers also specify:
  - **partition** (k_2, number of partitions) → partition for k_2
    - Often a simple hash of the key, e.g., hash(k_2) mod n
    - Divides up key space for parallel reduce operations
  - **combine** $(k_2, [v_2]) \rightarrow [(k_3, v_3)]$
    - Mini-reducers that run in memory after the map phase
    - Used as an optimization to reduce network traffic



Two more details…

- Barrier between map and reduce phases
  - But we can begin copying intermediate data earlier
- Keys arrive at each reducer in sorted order
  - No enforced ordering *across* reducers

"Hello World": Word Count

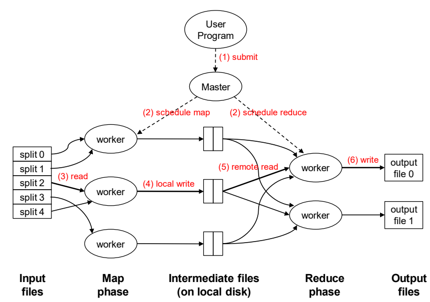
```
Map(String docid, String text):
    for each word w in text:
        Emit(w, 1);

Reduce(String term, Iterator<Int> values):
    int sum = 0;
    for each v in values:
        sum += v;
        Emit(term, value);
```


MapReduce can refer to…

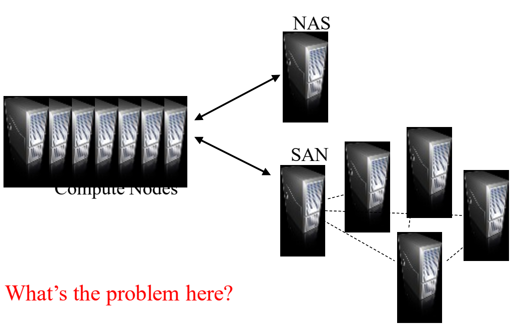
- The programming model
- The execution framework (aka "runtime")
- The specific implementation

MapReduce Implementations

- Google has a proprietary implementation in C++
  - Bindings in Java, Python
- Hadoop is an open-source implementation in Java
  - Development led by Yahoo, used in production
  - Now an Apache project
  - Rapidly expanding software ecosystem
- Lots of custom research implementations
  - For GPUs, cell processors, etc.



How do we get data to the workers?



What's the problem here?

Distributed File System

- Don't move data to workers… move workers to the data!
  - Store data on the local disks of nodes in the cluster
  - Start up the workers on the node that has the data local
- Why?
  - Not enough RAM to hold all the data in memory
  - Disk access is slow, but disk throughput is reasonable
- A distributed file system is the answer
  - GFS (Google File System) for Google's MapReduce
  - HDFS (Hadoop Distributed File System) for Hadoop

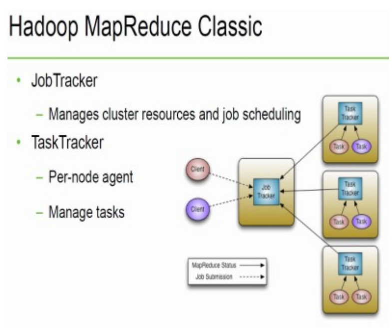
Recap

- Why large data?
- Cloud computing and MapReduce
- Large-data processing: “big ideas”
- What is MapReduce?
- Importance of the underlying distributed file system

Hadoop 2.0 and YARN

YARN

- Yet Another Resource Negotiator
- YARN Application Resource Negotiator (Recursive Acronym)
- Remedies the scalability shortcomings of “classic” MapReduce
- Is more of a general purpose framework of which classic mapreduce is one application.

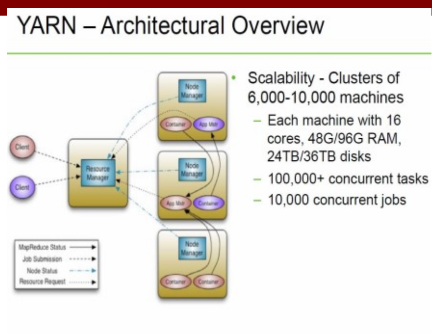


MapReduce Limitations

- Scalability
  - Maximum Cluster Size – 4000 Nodes
  - Maximum Concurrent Tasks – 40000
  - Coarse synchronization in Job Tracker
- Single point of failure
  - Failure kills all queued and running jobs
  - Jobs need to be resubmitted by users
- Restart is very tricky due to complex state

YARN

- Splits up the two major functions of JobTracker
  - Global Resource Manager - Cluster resource management
  - Application Master - Job scheduling and monitoring (one per application). The Application Master negotiates resource containers from the Scheduler, tracking their status and monitoring for progress. Application Master itself runs as a normal *container*.
- Tasktracker
  - NodeManager (NM) - A new per-node slave is responsible for launching the applications’ containers, monitoring their resource usage (cpu, memory, disk, network) and reporting to the Resource Manager.
- YARN maintains compatibility with existing MapReduce applications and users.



Classic MapReduce vs. YARN

- ❖Fault Tolerance and Availability
  - ❖Resource Manager
    - ❖ No single point of failure – state saved in ZooKeeper
    - ❖ Application Masters are restarted automatically on RM restart
  - ❖Application Master
    - ❖Optional failover via application-specific checkpoint
    - ❖MapReduce applications pick up where they left off via state saved in HDFS
- ❖Wire Compatibility
  - ❖Protocols are wire-compatible
  - ❖Old clients can talk to new servers
  - ❖Rolling upgrades

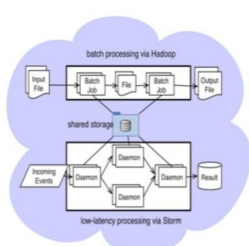
Classic MapReduce vs. YARN

- ❖Support for programming paradigms other than MapReduce (Multi tenancy)
  - ❖Tez – Generic framework to run a complex DAG
  - ❖HBase on YARN(HOYA)
  - ❖Machine Learning: Spark
  - ❖Graph processing: Giraph
  - ❖Real-time processing: Storm
  - ❖Enabled by allowing the use of paradigm-specific application master
  - ❖*Run all on the same Hadoop cluster!*

Storm on YARN

- ❖Motivations
  - ❖Collocating real-time processing with batch processing
  - ❖Provides a huge potential for elasticity.
  - ❖Reduces network transfer rates by moving storm closer to Mapreduce.

Storm on YARN @Yahoo



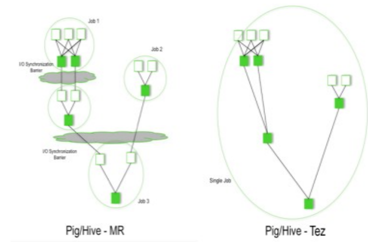
Storm on YARN @Yahoo

- ❖Yahoo enhanced Storm to support Hadoop style security mechanisms
- ❖Storm is being integrated into Hadoop YARN for resource management.
- ❖Storm-on-YARN enables Storm applications to utilize the computational resources in our tens of thousands of Hadoop computation nodes.
- ❖YARN is used to launch the Storm application master (Nimbus) on demand, and enables Nimbus to request resources for Storm application slaves (Supervisors).

Tez on YARN

- Hindi for speed
- Currently in development
- Provides a general-purpose, highly customizable framework that creates simplifies data-processing tasks across both small scale (low-latency) and large-scale (high throughput) workloads in Hadoop.
- Generalizes the MapReduce paradigm to a more powerful framework by providing the ability to execute a complex DAG
- Enables Apache Hive, Apache Pig and Cascading can meet requirements for human-interactive response times and extreme throughput at petabyte scale.

Tez on YARN



Tez on YARN

- Performance gains over Mapreduce
  - Eliminates replicated write barrier between successive computations
  - Eliminates job launch overhead of workflow jobs
  - Eliminates extra stage of map reads in every workflow job
  - Eliminates queue and resource contention suffered by workflow jobs that are started after a predecessor job completes

Tez on YARN

- Is part of the Stinger Initiative
- Should be deployed as part of Phase 2

HBase on YARN(HOYA)

- Currently in prototype
- Be able to create on-demand HBase clusters easily - by and or in apps
  - With different versions of HBase potentially (for testing etc.)
- Be able to configure different HBase instances differently
  - For example, different configs for read/write workload instances
- Better isolation
  - Run arbitrary co-processors in user's private cluster
  - User will own the data that the HBase daemons create

HBase on YARN(HOYA)

- MR jobs should find it simple to create (transient) HBase clusters
  - For Map-side joins where table data is all in HBase, for example
- Elasticity of clusters for analytic / batch workload processing
  - Stop / Suspend / Resume clusters as needed
  - Expand / shrink clusters as needed
- Be able to utilize cluster resources better
  - Run MR jobs while maintaining HBase's low latency SLAs

Classic MapReduce vs. YARN

- ❖Multi-tenancy, being able to run multiple paradigms simultaneously is a big plus.

Hadoop 2.0

- ❖So Hadoop 2.0 includes YARN, High Availability and Federation
- ❖High Availability takes away the Single Point of failure from namenode and introduces the concept of the QuorumJournalNodes to sync edit logs between active and standby namenodes
- ❖Federation allows multiple independent namespaces(private namespaces, or hadoop as a service)